REAL-TIME ANIMATION AND RENDERING OF OCEAN WAVES

Joel Barrett

BSc (Hons) Computer Games Technology

2012

University of Abertay Dundee
Institute of Arts, Media and Computer Games

Declaration of Originality and Permission to Copy

Author:	Joel Barrett
Title:	Real-time Animation and Rendering of Ocean Waves
Degree sought:	BSc (Hons) Computer Games Technology
Year of Submission:	2012

i. *I certify that the above mentioned project is my original work.*

ii. *I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.*

Signature: ……

Date: ……..…..……

Abstract

Fluid simulation is an important area of research in computer graphics, given the prevalence of fluid in the natural world. This project aims to address some of the key issues involved in simulating ocean surface waves within a game. An experimental framework is developed in order to evaluate an approach by Jerry Tessendorf, which generates sinusoidal waves on a heightmap using an inverse fast Fourier transform and statistics gathered from oceanographic research. The surface is shaded by taking into account scattered light, reflection of the sky, Fresnel term and specular highlights from the sun. A final rendering of the surface is presented in juxtaposition to a photograph of a real ocean, and the various stages of the algorithm are profiled, with experiments conducted to determine the effects of wind velocity and FFT size on the simulation. Next, a comparison is drawn between two methods of computing normals – that of the FFT and Sobel filter. In conclusion, it is found that the approach provides a fairly realistic simulation of ocean waves on deep water that is suitable for use within a real-time environment such as a game.

Acknowledgements

I would like to thank my project supervisor, Dr Matthew Craven and module tutor, Dr Henry Fortuna for their guidance during this process. I would also like to extend my appreciations to the various lecturers who have taught me over the years. Finally, I owe a debt of gratitude to my parents for being pillars of support throughout my undergraduate education.

– Joel Barrett, 2012

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Water (H_2O) covers roughly 71% of the Earth's surface, and is critical to all known forms of life. From municipal plumbing to recreation, food processing to irrigation, it is ubiquitous in everyday life, and as such, is studied across a range of fields. One of these fields is computer graphics, where photorealistic rendering – or simulating the visual world around us as closely as possible – is often a key goal (Foley et al. 1995).

Fluid simulation can lead to some of the most striking animations in computer graphics, such as the bubbling, foaming and splashing of liquids, and the swirling of gases. However, modelling the various nuances and forms of fluid in a convincing manner is an on-going challenge for practitioners, with a key balancing act being that of computational complexity versus hardware constraints. Furthermore, the various forms of fluid require bespoke simulation approaches – for example, modelling a fast-flowing river requires a different methodology to modelling a droplet. Thus, for this project to be feasible within its time constrictions, it was necessary to focus on a particular type of fluid – water – and a particular scale – oceans.

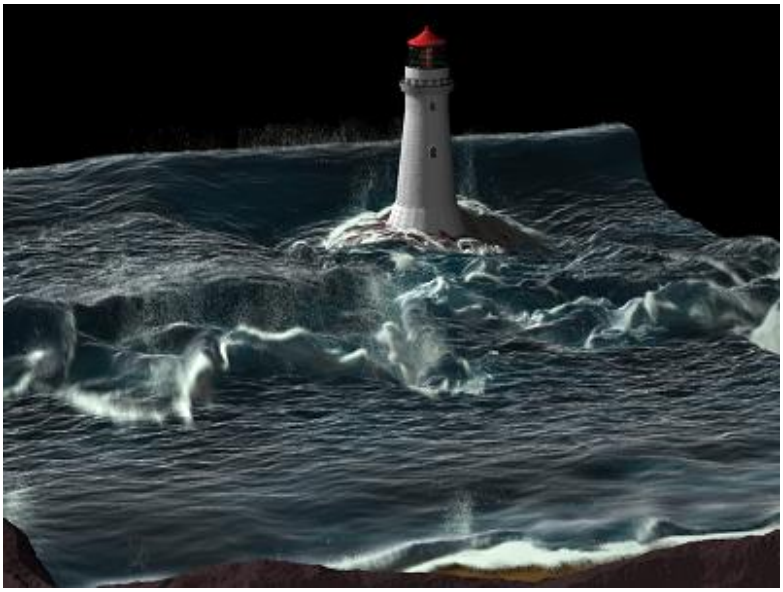

Figure 1: Two-way coupled SPH and particle level set fluid simulation (Losasso et al. 2008).

Oceans exhibit highly dynamic behaviour, ranging from minor turbulent waves to enormous shore breaks. The state of the ocean is altered by a multitude of phenomena occurring at small and large scales. In oceanographic literature, the behaviour of an ocean's surface is generally defined by two locations: deep areas – those far away from the coast – and intermediate or shallow areas – those near to the shore (Darles et al. 2011). A key differentiator is that in deep water, the effect of the ocean floor is negligible, so wave shoaling and refraction may be ignored.

1.1 Aim and Objectives

The aim of this work is to investigate ocean simulation in games, with a view to answering the following question:

"How can procedurally animated, realistically shaded ocean surface waves be synthesised in a real-time environment as they pass across deep water?"

Consideration will be given as to how this complex process can be simplified for use in a real-time environment, without overly compromising the essence of what makes the ocean appear as it does. In undertaking this project, it is hoped that the following objectives will be achieved:

- To research and outline current approaches to ocean water simulation.
- To construct an experimental framework to implement the research discoveries and gather quantitative results.
- To determine the effectiveness of the implementation by analysis and evaluation of the results.
- To draw conclusions and make recommendations for future work.

1.2 Organisation of this Dissertation

Firstly, Chapter 2 introduces some fundamental concepts of fluid dynamics and contextualises this work with a historical perspective of water simulation in games. Chapter 3 then presents research in fluid simulation for computer graphics, which dates back to the early 1980's. Based on the findings of this review, Chapter 4 describes the implementation of Tessendorf's model of ocean simulation. Next, Chapter 5 presents results from experiments that were undertaken and discusses their implications. Lastly, Chapter 6 draws this dissertation to a conclusion and recommends future work that could be undertaken.

Chapter 2

Background

Some prior knowledge is expected of the reader of this dissertation – namely, a familiarity with computer graphics, Newtonian mechanics and basic vector calculus like the differential operators gradient (∇), divergence ($\nabla\cdot$) and curl ($\nabla\times$). Coverage of these subjects may be found in Foley et al. (1995), French (1971) and Matthews (1998), respectively.

2.1 Incompressible Navier-Stokes Equations

Most fluid flow in computer graphics is governed by the incompressible Navier-Stokes equations. These are a collection of partial differential equations that should hold throughout the body of the fluid, and are commonly expressed as:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} + \frac{1}{\rho}\nabla p = \boldsymbol{g} + \upsilon\nabla\cdot\nabla\boldsymbol{u}, \qquad (1)$$

$$\nabla\cdot\boldsymbol{u} = 0. \qquad (2)$$

Where $\boldsymbol{u}$ is the velocity of the fluid; ρ is density; p is pressure, the force per unit area exerted by the fluid on something; $\boldsymbol{g}$ is acceleration caused by gravity and υ is

kinematic viscosity, a measure of how much the fluid resists deformation as it flows (or, as Bridson (2008) explains: “how difficult it is to stir”).

Equation 1 is known as the momentum equation, and is derived from Newton’s 2nd law of motion, $\boldsymbol{F} = m\boldsymbol{a}$. It gives us the acceleration of a fluid due to the forces acting on it. Equation 2 is the incompressibility condition; it specifies that the volume of the fluid will remain constant. Changes to volume – such as those caused by underwater acoustics – are usually negligible, and not worth considering in computer graphics, where appearance is the main consideration, not accuracy.

2.2 Eulerian and Lagrangian Specifications of the Flow Field

There are two frames of reference in fluid dynamics: Eulerian and Lagrangian. The Eulerian viewpoint considers how values like velocity and density change with time at fixed points in space. This can be visualized by sitting on a river bank and watching the water flow past a fixed location. The Lagrangian viewpoint, on the other hand, tracks individual fluid parcels as they move through space and time. This is synonymous to the concept of a particle system, and is more in accordance to the molecular composition of fluid in nature. It can be visualized by sitting in a boat as it drifts down a river. In computer graphics, Eulerian fluid is generally arranged into heightmaps or voxels, and Lagrangian fluid is often represented by an implicit surface with particles for control points.

2.3 Physically-based and Oceanographic Approaches

Within the sub-discipline of fluid simulation concerned with oceans, techniques can be categorised as physically-based – those governed by the incompressible Navier-Stokes equations, suitable for shallow water – or spatial/spectral – those based on oceanographic research, suitable for deep water. Spatial methods represent an ocean surface as a heightmap composed of sums of periodical functions, where animation is produced by taking phase differentials. Spectral methods represent an ocean surface as a wave spectrum and apply a Fourier transform to convert it from the spectral domain to the spatial domain. Hybrid models that combine spatial and spectral characteristics also exist.

2.4 Water Simulation in Games

Early implementations of water during the rise of 3D gaming in the nineties treated the body of water as a uniformly coloured planar surface. With the first application of texture mapping to games, titles like *Ultima Underworld: The Stygian Abyss* (1992) and *Doom* (1993) added surface detail akin to the differences in shading that waves propagating along the surface would create. Alpha blending was later added to emulate the transparency of water, and motion could be achieved by cycling through textures in a texture atlas, which were often simply images of water caustics that could be looped seamlessly. The scene in Figure 2 shows these techniques in action.

Figure 2: Alpha-blended water plane in Tomb Raider II (1997).

These methods were sufficient to portray a body of water to the player in a crude way, and could be extended to provide some interactivity. Decals, billboards and particles could be spawned to represent splashes when an object intersected the water plane, and the player character could be permitted to swim underwater by removing the burden of gravity when below the water plane. Later, with the advent of the stencil buffer, planar reflections could be efficiently added to the surface in order to reflect the surrounding environment.

In the latter part of the nineties, titles like *Wave Race 64* (1996) lead the way in replacing the static water plane with a heightmap capable of propagating geometric waves along its surface. This gave rise to the first physically-based methods, such as the central difference approximation of the two-dimensional wave equation, which is commonly used by games to model capillary waves (Gomez 2000).

Chapter 3

Literature Review

Early exploration into fluid simulation for computer graphics began in the 1980's. Initial methods applied bump mapping to perturb normals in different ways, such as Schachter (1980), who transformed normals using a sum of twenty cycloids. Max (1981) introduced a hydrodynamic model that computed a heightmap from a series of low amplitude sine waves. The frequencies were carefully selected to repeat at a common point in time so that the animation could be looped seamlessly. However, this created a caveat in that the pattern of waves generated could be seen to repeat itself when viewed from afar. In both of these methods, the ocean floor is treated as infinite, so wave shoaling and refraction were not possible as this point in time.

At the ACM SIGGRAPH conference in 1986, two new papers were accepted on ocean water simulation. Peachey (1986) applied Airy wave theory to compute waves that took depth into account for the first time, which meant that breaking waves and refraction could now be simulated. He also used depth to conditionally emit particles for spray, and to apply a quadratic function to basic sinusoids, which gave them a more realistic choppy appearance. In the other paper, Fournier and Reeves (1986)

proposed various modifications to Gerstner's wave theory that considered the topology of the ocean floor when calculating the path of water particles, and altered their normal circular path to be elliptic. This meant that the shape of the wave crest could be modified in order to potentially produce a more realistic result.

The first physically-based approach to ocean water simulation was suggested by Kass and Miller (1990). Their method involved approximating the shallow water – or Saint-Venant – equations in two dimensions, and applying the result to a heightmap. Other simplifications were suggested such as ignoring the vertical component of water particle velocity and treating horizontal velocity in heightmap columns as roughly constant. Despite these simplifications, the method was able to consider wave refraction with depth, reflecting waves, net water transport and scenarios where boundaries vary over time. Chen and Lobo (1995) later extended this approach to solve the Navier-Stokes equations with a pressure term, thus providing a more realistic model of the ocean.

A seminal paper was later written by Stam (1999) in which he introduced an unconditionally stable technique. This turned out to be the key to simulating complex fluid-like behaviour in real-time, as with stability, larger time steps can be taken. This meant that more efficient animation was possible with comparable realism to unstable schemes. The method of Foster and Metaxas (1996) was used as a starting point, but instead of being an explicit Eulerian solver, a combination of Lagrangian and implicit methods was proposed. This approach was later implemented on the GPU by Harris (2004). However, Stam's method is not applicable to this project as it does not address

free surfaces like the ocean, and the animation produced is more akin to the movement of gaseous phenomena (Iglesias 2004).

The key paper that this project will seek to investigate was written by Tessendorf (2001), and is based on the work of Mastin, Watterberg and Mareda (1987), who introduced the first spectral approach to ocean simulation. In it, he describes an Eulerian method that utilises empirical data. The ocean is considered incompressible, irrotational (i.e. curl is zero) and inviscid (i.e. no viscosity). Rather than solving the full non-linear Navier-Stokes equations, motion of the surface is restricted to potential flow, which permits the Navier-Stokes equations to be simplified into Bernoulli's equation. Wave height is considered an arbitrary variable of horizontal position and time. The corresponding heightmap is decomposed into a series of sinusoidal waves and evaluated with an inverse fast Fourier transform. Random sets of amplitudes are created in a way that adheres to oceanographic phenomenology, and the Phillips spectrum is used to model the effect of wind as it blows the waves. A variety of enhancements are proposed to the Phillips spectrum that permit the direction, speed and parallel disposition of the resulting waves to be tuned. Tessendorf also suggests post-processing the heightmap to increase the choppiness of waves, as the FFT does not produce the trochoidal shapes that waves exhibit in nature.

With regard to industrial application of Tessendorf's method, there have been a few papers written by game developers who are actively using the technique. Mittring (2007) describes how Crytek applied LOD and screen-space tessellation to their implementation of Tessendorf's work in *CryEngine 2*. Day (2009) writes that Insomniac used the technique to simulate water in *Resistance 2*. The technique has

also been used significantly within the special effects industry, but via a more accurate approach. The key to the technique's scalability is that the size of the heightmap can be altered. Within the games industry, Insomniac used 32 x 32 heightmaps, and Crytek, 64 x 64, which is in contrast to the 2056 x 2056 heightmaps used for the offline simulations in movies like *Waterworld* and *Titanic* (Tessendorf 2001).

Recently in the field, Yuksel (2010) introduced a new technique called "wave particles" that permits fast, unconditionally stable solid-fluid coupling by tracking surface deviations from waves. This method can be applied to a large-scale body of fluid like an ocean, with reasonable performance for a number of interactions. An alternative to this is to solve the 2D wave equation using finite differences, as proposed by Cords and Staadt (2009). The finite differences approach scales better than wave particles when there are lots of disturbances because the grid will have a constant number of squares to evaluate at each cycle, regardless of wave activity, whereas wave particles would need to be spawned for each disturbance. Using the wave equation also permits Huygen's principle, which means that waves can be diffracted and the richness of the animation consequently improved. Tessendorf's method is used in both articles to generate ambient (i.e. non-interactive) waves.

Chapter 4

Methodology

From the literature review, it is clear that Tessendorf's method is a promising technique for ocean simulation in games, as it can be scaled down to meet the requirements of real-time simulation, and has seen commercial application. Thus, its implementation in DirectX 11 will now be discussed in this chapter.

4.1 Representing the Ocean Surface

The application makes use of an Eulerian framework, so the ocean surface is represented by a heightmap. Each vertex in the heightmap is a custom vertex format with 3-component position and normal. The following function is used to generate a 1D array of vertices in the x-z plane, centred about the origin:

```
int GenerateVertices(VertexPosNor** vertices, int dimensions, float stride)
{
    int numVertices = dimensions * dimensions;
    *vertices = new VertexPosNor[numVertices];

    float halfDim = (dimensions - 1.0f) / 2.0f;

    for (int z = 0; z < dimensions; ++z)
    {
        for (int x = 0; x < dimensions; ++x)
```

```
        {
            (*vertices)[z * dimensions + x] = VertexPosNor((x - halfDim) *
                stride, 0.0f, (z - halfDim) * stride, 0.0f, 1.0f, 0.0f);
        }
    }
    return numVertices;
}
```

For efficiency, the generated vertices are stored in a vertex buffer. Indices are then computed and written to an index buffer, with the mesh rendered via an invocation of DrawIndexed().

4.2 Initialising the Ocean Surface

The first stage of the simulation is initialisation of the heightmap. The algorithm is concerned with synthesising a fully developed ocean – that is, one where the wind has been affecting it for some time and waves have reached their peak amplitudes. Therefore, the heightmap will be instantiated with an initial series of heights, rather than being flat. The amplitudes of the wave heightmap can be found by evaluating:

$$\tilde{h}_0(\boldsymbol{k}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\boldsymbol{k})}. \qquad (3)$$

Where $\boldsymbol{k}$ is the two-dimensional wave vector $(\frac{2\pi M}{L_x}, \frac{2\pi N}{L_z})$ pointing in the direction of travel of the wave, M and N are the dimensions of the heightmap, L_x and L_z are the dimensions of the ocean patch, ξ_r and ξ_i are pseudo-random numbers drawn from a probability distribution, and $P_h(\boldsymbol{k})$ is an ocean wave spectrum. The remainder of this section will describe the schemes used to compute the pseudo-random numbers and ocean wave spectrum.

4.2.1 Pseudo-random Number Sampling

There are many different probability distributions available, but Tessendorf (2001) recommends use of the normal (Gaussian) distribution for random number generation, as it "tends to follow the experimental data on ocean waves". He suggests computing ξ_r and ξ_i via two Gaussian random numbers with mean 0 and standard deviation 1 – otherwise known as a "standard normal" distribution (Thomas et al. 2007). This has the probability density function:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \quad (4)$$

Which, when plotted $\phi(x)$ versus x, generates a bell-curve shape. There are various different algorithms for generating Gaussian random numbers, such as the Box-Muller transform, Marsaglia polar method and Ziggurat algorithm. Performance is often a deciding factor when selecting an algorithm to use, but in this instance, random numbers are only required when initialising the ocean surface, so speed is not an important factor. As such, the Box-Muller transform was chosen, as it is straightforward to implement. It produces a pair of Gaussian random numbers that represent coordinates on a 2D plane, and was implemented as such:

```
float GaussRand()
{
    // Uniform random numbers
    float u1 = rand() / (float)RAND_MAX;
    float u2 = rand() / (float)RAND_MAX;

    // Box-Muller transform
    if (u1 < 1e-6f) {
        u1 = 1e-6f;
    }
    return sqrtf(-2.0f * logf(u1)) * cosf(XM_2PI * u2);
}
```

4.2.2 Ocean Wave Spectra

There are various spectra available for describing the state of the ocean as it is affected by the wind, such as the Pierson-Moskowitz, JONSWAP and TMA spectrums. Tessendorf (2001) recommends use of the Phillips spectrum, which can be used to synthesise wind-generated waves larger than capillary waves in a fully developed ocean, and is defined as:

$$P_h(\boldsymbol{k}) = A\frac{e^{\frac{-1}{\|\boldsymbol{k}\|^2 L^2}}}{\|\boldsymbol{k}\|^4}\left|\hat{\boldsymbol{k}}\cdot\hat{w}\right|^2. \quad (5)$$

Where A is a numeric constant that can be used to scale global wave amplitude, $\|\boldsymbol{k}\|$ is the magnitude of the wave vector $\boldsymbol{k}$, $L = V^2/g$ is the largest possible wave to arise from a constant wind speed V, g is acceleration due to gravity, and $\hat{w}$ is the normalised direction of the wind. $\left|\hat{\boldsymbol{k}}\cdot\hat{w}\right|^2$ is a cosine factor that eliminates waves moving perpendicular to the wind direction. In this form, Phillips spectrum produces waves that move both with and against the wind, resulting in meeting waves that could be used to simulate foaming and splashing. When the dot product in the cosine factor is negative, the wave will move against the wind, so the absolute value of this could be taken in order to dampen out waves moving against the wind. The convergence properties of the spectrum can be improved by filtering out waves with very small length l. This can be achieved by adding the multiplicative factor $e^{-\|\boldsymbol{k}\|^2 l^2}$ to Equation 5. Both of these modifications were applied, which resulted in the following code:

```
float Ocean::Phillips(const XMFLOAT2 &k)
{
    if (k.x == 0.0f && k.y == 0.0f) {
        return 0.0f;
```

```
    }
    // Largest possible wave from constant wind speed V
    float L = (settings_.V * settings_.V) / gravity_;

    // Smallest possible wave from constant wind speed V
    float l = L / settings_.smallestWave;

    float ksqr = k.x * k.x + k.y * k.y;
    float hcosf = k.x * cosf(settings_.w) + k.y * sinf(settings_.w);
    float retval = settings_.A * (expf(-1.0f / (ksqr * L * L)) /
        (ksqr * ksqr * ksqr)) * (hcosf * hcosf);

    // Filter out waves moving opposite to wind
    if (hcosf < 0.0f) {
        retval *= settings_.S;
    }
    return retval * expf(-ksqr * l * l);
}
```

4.3 Animating the Ocean Surface

The next step is to compute the Fourier amplitudes of the wave heightmap at time t. The following equation will be evaluated at each cycle of the algorithm:

$$\tilde{h}(\boldsymbol{k}, t) = \tilde{h}_0(\boldsymbol{k}) e^{i\omega(\|\boldsymbol{k}\|)t} + \tilde{h}_0^*(-\boldsymbol{k}) e^{-i\omega(\|\boldsymbol{k}\|)t}. \qquad (6)$$

Where ω is the angular frequency of wave $\boldsymbol{k}$ (i.e. the speed at which the wave travels across the surface), t is time, and * is the conjugate of a complex number. As $\tilde{h}_0(\boldsymbol{k})$ is computed at initialisation, the remaining unknown to determine is angular frequency. In deep water, where the effect of the ocean floor may be ignored, the relationship between ω and $\boldsymbol{k}$ can be expressed as:

$$\omega(\|\boldsymbol{k}\|) = \sqrt{g\|\boldsymbol{k}\|}. \qquad (7)$$

For efficiency, this equation was evaluated at each grid point as the ocean surface is initialised, which prevents additional computation during the update cycle, in particular the expensive square root operation. As mentioned, Equation 6 produces a

series of Fourier amplitudes – the next step is to convert these into a series of height values in the heightmap.

4.3.1 Fourier Transform

The Fourier transform was first described by French mathematician and physicist Joseph Fourier in 1811, and provides a means of decomposing an input signal into its constituent frequencies. Today, it has seen applications ranging from data compression to magnetic resonance imaging (MRI). There are three variants of the Fourier transform: the standard Fourier transform, the Fourier series and the discrete Fourier transform (DFT). The last of these is responsible for a recent proliferation of applications in digital signal processing, following the discovery of the fast Fourier transform (FFT).

4.3.1.1 Fast Fourier Transform

The fast Fourier transform provides a means of efficiently calculating DFTs, and is commonly attributed to Cooley and Tukey[1] (1965). The FFT is used to convert a function from the spatial domain to the frequency domain – or vice versa in the case of its inverse – in $O(n \log n)$ time. Implementing an efficient FFT on a computer is a significant undertaking, so implementing one from scratch was considered out with the scope of this project. As such, the popular Fastest Fourier Transform in the West (FFTW) library was used, which provides a thoroughly optimised FFT implementation on the CPU.

[1] In fact, it was recently discovered that Carl Friedrich Gauss invented it around 1806 in a paper that was published posthumously.

In this scenario, an inverse FFT is used to efficiently evaluate the sum of a set of sine waves with various different amplitudes and phases, in order to compute heights in the heightmap. This involves finding the wave height $h(\boldsymbol{p}, t)$, where $\boldsymbol{p}$ is the horizontal coordinate (x, z) on the heightmap. This can be found by solving the following equation:

$$h(\boldsymbol{p}, t) = \sum_{\boldsymbol{k}} \tilde{h}(\boldsymbol{k}, t) e^{i\boldsymbol{k}\cdot\boldsymbol{p}}. \tag{8}$$

Where $\tilde{h}(\boldsymbol{k}, t)$ is the Fourier amplitude found by solving Equation 6. Evaluating this equation at each cycle of the application produces an animated ocean surface.

4.3.2 Choppy Wave Modification

The FFT produces waves with rounded peaks and troughs, but waves in the ocean tend to have trochoidal shapes (i.e. thin peaks and flattened troughs). Hence, post-processing of the ocean heightmap is required in order to produce wave profiles more akin to the choppy surface in Figure 3 below.

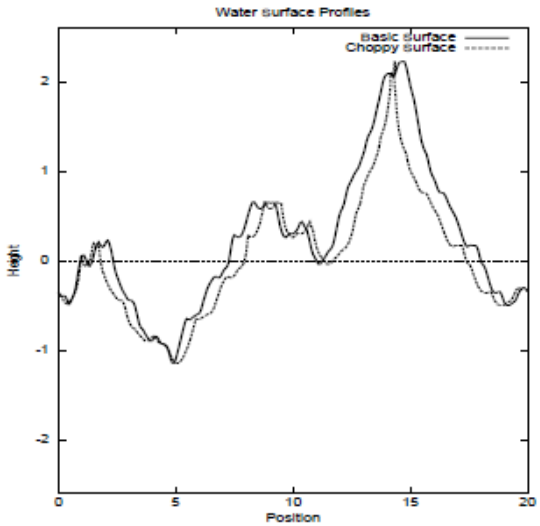

Figure 3: Comparison of choppy and standard wave profiles (Tessendorf 2001).

To achieve this shape, the position of each grid point $\boldsymbol{p}$ should be displaced horizontally by the factor:

$$\boldsymbol{p} = \boldsymbol{p} + \lambda \boldsymbol{d}(\boldsymbol{p}, t). \tag{9}$$

Where λ is a constant for setting the degree of displacement, and $\boldsymbol{d}$ is a two-dimensional displacement vector (x, z) – the components of which can be computed by evaluating the inverse FFT:

$$\boldsymbol{d}(\boldsymbol{p}, t) = \sum_{\boldsymbol{k}} -i \frac{\boldsymbol{k}}{\|\boldsymbol{k}\|} \tilde{h}(\boldsymbol{k}, t) e^{i\boldsymbol{k}\cdot\boldsymbol{x}}. \tag{10}$$

So, the displaced position $\boldsymbol{p}'$ of a point $\boldsymbol{p} = (x, y, z)$ on the heightmap at time t can be summarised as:

$$\boldsymbol{p}'^{(t)} = \begin{pmatrix} x'(t) \\ y'(t) \\ z'(t) \end{pmatrix} = \begin{pmatrix} x + \lambda \boldsymbol{d}(\boldsymbol{p}, t) \\ h(\boldsymbol{p}, t) \\ z + \lambda \boldsymbol{d}(\boldsymbol{p}, t) \end{pmatrix}.$$

4.4 Rendering the Ocean Surface

Now that the dynamics of the ocean surface have been covered, lighting and colouring must be handled, for the details of wave shape would be lost with smooth-shading.

4.4.1 Computing Normals

Before the surface can be shaded, normals must be computed from the heightmap. Tessendorf (2001) mentions two ways to compute normals: with additional FFTs or a

finite difference scheme. The latter is disregarded by Tessendorf due to its poor approximation of slope for waves with small wavelength; but, for the sake of juxtaposition, both the FFT method and a Sobel filter method were implemented.

4.4.1.1 Fast Fourier Transform Normals

An exact computation of the slope vector can be obtained using a further two FFTs – that is, by evaluating the equation:

$$\nabla h(\boldsymbol{p}, t) = \sum_{\boldsymbol{k}} i\boldsymbol{k}\tilde{h}(\boldsymbol{k}, t)e^{i\boldsymbol{k}\cdot\boldsymbol{p}}. \tag{11}$$

4.4.1.2 Sobel Filter Normals

Another way to compute normals is via the Sobel operator, which is commonly used in image processing for edge detection. The operator makes use of the following 3 x 3 kernels – one vertical, the other horizontal – to convolve an image in order to find approximate derivatives:

$$G_x = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} * A, \qquad G_y = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix} * A.$$

Where G_x and G_y are images that contain vertical and horizontal derivative approximations, A is the source image, and $*$ denotes the two-dimensional convolution operation. So, to compute the normal at a grid point, the heights of the neighbouring orthogonal and diagonal grid points are sampled using these kernels.

4.4.2 Colouring the Ocean Surface

In nature, most oceans appear as a dark navy blue, with the blue colour being caused by the absorption and scattering of light, and the darkness being caused by depth (i.e. there is no reflection of light from the sea floor). However, areas of ocean with a high concentration of phytoplankton can appear more green in colour, due to the presence of chlorophyll in their cells (NASA [no date]). In this application, the colouring scheme of Jensen and Goliáš (2001) is used, whereby the water appears dark blue when looking into its depth, and light green when looking at the waves.

4.4.3 Reflection

The reflection vector R can be computed as:

$$\mathrm{R} = \mathrm{I} - 2\mathrm{N}(\mathrm{I} \cdot \mathrm{N}). \qquad (12)$$

Where I is the incident vector and N is the normal vector. The application takes the camera view to be I, and uses the resultant 3D reflection vector to sample the cube texture used for the skybox, in order to simulate reflection of the sky in the water.

4.4.4 Fresnel Term

Reflective surfaces become increasingly reflective as the viewing angle approaches a grazing angle (that is, parallel to the surface). This can be described with the Fresnel term, which is almost always expressed in the computer graphics community using an approximation by Schlick (1994), in which the specular reflection coefficient R is given by:

$$\mathrm{R}(\theta) = \mathrm{R}_0 + (1 - \mathrm{R}_0)(1 - \cos(\theta)). \qquad (13)$$

Where θ is the half angle between the incoming and outgoing light directions, and R_0 is the reflectance at normal incidence - i.e. the value of the Fresnel term when $\theta = 0$ (Wikipedia 2012). The reflectivity of the surface is then set by linearly interpolating between the water colour and skybox texture, based on the value of the Fresnel term.

4.4.5 Specular Light from the Sun

Finally, light produced by the sun is added to the surface. This is just treated as a generic specular highlight that takes into account the direction of the sun, the reflection vector that was previously calculated, the shininess of the water surface (to widen or narrow the specular highlight), and the colour of the light emitted from the sun. Gathering the various techniques described in this section results in the following pixel shader that is applied to the ocean surface:

```
float4 OceanSolidPS(PS_INPUT input) : SV_Target
{
    // Compute view and blend between two colours to simulate scattered light
    float3 view = normalize(camPos - input.wsPos);
    float4 waterColour = lerp(shallowWaterColour, deepWaterColour,
        saturate(dot(view, input.normal)));

    // Reflect view through normal and sample the environment map
    float3 reflectivity = reflect(-view, input.normal);
    float4 skyReflection = textureSkyReflection.Sample(samplerSkyReflection,
        reflectivity);

    // Blend the water and sky reflection colours using Fresnel term
    float fresnel = 0.0204f + 0.9796f * pow(saturate(1.0f - dot(I, N)), 5.0f);
    float4 fragmentColour = lerp(waterColour, skyReflection, fresnel);

    // Add specular light from the sun
    float4 sunSpecular = pow(saturate(dot(reflectivity,
        normalize(sunDirection))), waterShininess) * sunColour;
    fragmentColour += sunSpecular;

    return fragmentColour;
}
```

Chapter 5

Results and Discussion

The previous chapter detailed the issues involved in implementing Tessendorf's method in a real-time environment. This chapter will now analyse that implementation, with a view to determining its effectiveness. The results presented in this section were produced with a laptop of the following specifications:

CPU	Intel Core i7-2670QM (6 MB Cache, 2.20 GHz)
RAM	6 GB, DDR3
GPU	NVIDIA GeForce GT 555M, 2 GB
OS	Microsoft Windows 7, 64-bit

Table 1: Development, test and profiling environment.

Microsoft PIX was used for frame analysis, and QueryPerformanceCounter() for high-resolution profiling of code segments. All timing results presented in this section are to 3 decimal places, and averages taken across 10 frames, unless otherwise specified. The ocean simulation is initialised with the configurations listed in Appendix C. If a

configuration differs, it will be explicitly stated. All profiling was of course executed on a release mode build of the application. The application was also soak tested – i.e. left to run overnight – to ensure the simulation did not become erratic: it passed.

5.1 Visual Analysis

Firstly, we shall consider the appearance of the ocean surface that Chapter 4 produces. Ultimately, the benchmark for computer graphics is the natural world, so below is a juxtaposition of a real ocean and the simulation in calm weather conditions.

Figure 4: (L) Photo of the Atlantic Ocean (R) Screenshot of the simulated ocean.

Obviously the clouds and colouring in the two environments are different, amongst other things, but the simulated ocean reflects the sky colour and clouds, as in the real world, and appears to be fairly realistic.

5.1.1 Effect of Wind Velocity on the Simulation

Perhaps the most interesting parameter to alter is wind velocity, which can be used to recreate the Beaufort scale. Figure 5 below shows the state of the ocean at wind velocities of 0, 5, 10 and 15 mph, from top left to bottom right.

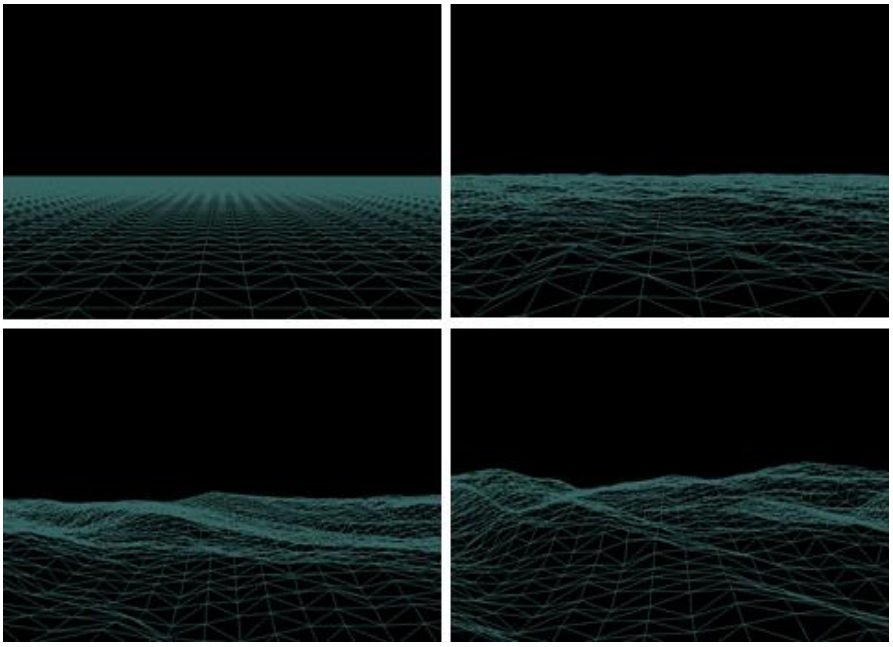

Figure 5: State of the ocean at various wind velocities.

5.2 Performance Analysis

In a video game, there are many other systems to process besides graphics, and with real-time processing and interactivity thrown into the equation, performance is important. The below tables show the results of profiling the algorithm. Firstly, Table 2 delineates the time spent processing the various components of the algorithm.

Task	**Time in milliseconds**
Initialise amplitudes and phases	13.067
Initialise 5 FFTW plans with FFTW_PATIENT[2]	3604.188
Update height values in the heightmap	2.688
Choppy waves modification	2.962

[2] This is a planner flag, which denotes the rigour applied by FFTW in selecting the fastest FFT algorithm to use in a given scenario. FFTW_PATIENT involves executing a wide range of different algorithms and comparing their performance characteristics.

Render the ocean	4.084

Table 2: Profiling results for the various components of the algorithm.

In the next table, the final application was timed with PIX to determine the FPS and milliseconds per frame of the application, averaged across 100 frames.

Mode	**Frames per second**	**Milliseconds per frame**
Windowed (800 x 600)	136.604	7.328
Fullscreen (1600 x 900)	98.398	10.215

Table 3: Framerate in windowed and fullscreen mode.

5.2.1 Effect of FFT Size on Performance

The below chart shows the time in milliseconds required to update the five 2D FFTs when they have various different two-power dimensions.

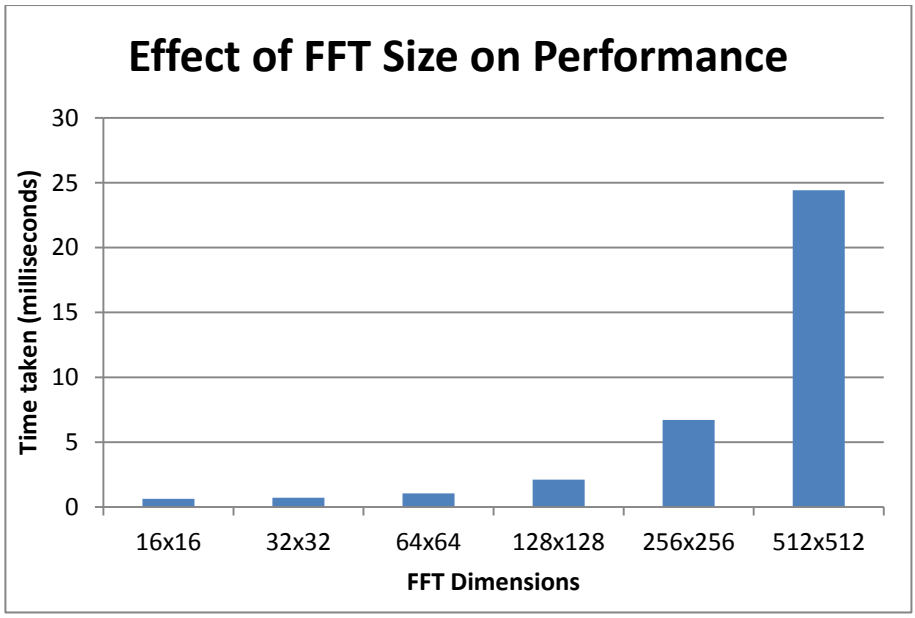

Figure 6: Time taken in milliseconds to update FFTs of various dimensions.

As can be seen, there is exponential growth in the amount of time required to process each FFT as the dimensions increase. The application makes use of a 256 x 256 FFT by default, as it provides a good level of detail without too much of a loss of performance.

5.3 Comparison of FFT and Sobel Filter Normals

As mentioned in Chapter 4, normals were implemented using both FFTs and a Sobel filter in order to compare the results. Figure 7 illustrates the visual difference between the techniques by rendering an identical scene. This was achieved by drawing the scene once just after initialisation with equivalent camera view and position, at the same point in time, and with an unseeded random number, so that Equation 3 will produce identical results.

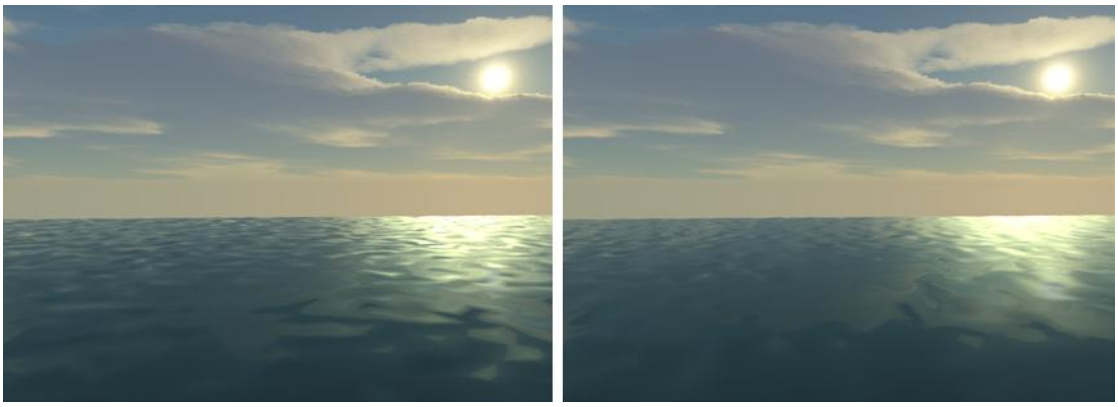

Figure 7: Normal vector methods (L) FFT normals (R) Sobel filter normals.

As expected, the analytical solution of the FFT produces more precise results as it is an exact calculation. The Sobel filter produces more blurred reflections, with some of the detail of the sky noticeably being lost in the bottom right-hand corner of the

image. The specular highlights from the sun in the Sobel filter image are more uniform, so that the shape of waves is lost near the centre where it is at its most intense. The shape of the specular highlights is more rounded in the FFT normals image, which is indicative of a higher resolution. In summary, the Sobel filter produces duller, flatter looking waves than the FFT solution, so using FFTs would be the recommended solution in terms of visual fidelity. Now we shall consider the performance impact of the two techniques. In both instances, normals are computed on the CPU and passed down the graphics pipeline to a pixel shader. Table 4 below shows the time in milliseconds each function takes to execute.

Function	**Time in milliseconds**
ComputeNormalsFFT()	1.447
ComputeNormalsSobel()	0.848

Table 4: Profiling results for FFT and Sobel filter normals.

So FFT normals provide greater accuracy, but at a greater cost. Thus, choice of normal scheme will depend on the requirements of the application.

Chapter 6

Conclusion

This project was undertaken in order to investigate ocean simulation in games, with a view to answering the question:

"How can procedurally animated, realistically shaded ocean surface waves be synthesised in a real-time environment as they pass across deep water?"

It has been demonstrated that the approach by Tessendorf (2001) provides a scalable ocean surface suitable for processing in a real-time environment. It is procedurally animated using FFTs, and realistically shaded by taking various environmental factors into account such as reflection of the sky and specular light from the sun. In constructing the application and this dissertation, the research question has been answered to some extent and the four objectives outlined in Chapter 1 were met.

6.1 Future Work

This project covers the core elements of Tessendorf's algorithm, but there are many other factors to consider in simulating an ocean. For instance, the application renders a

single patch of ocean waves, but to be of use in most scenes, this would need to be tiled to the horizon in order to create a seamless ocean surface. The FFT is periodic, so the heightmap can be tiled without modification. However, this repetition becomes distracting at a distance when numerous ocean patches are in view, as the pattern of waves can be seen to repeat itself. To eliminate this visual artefact, entropy can be added to the wave animation using Perlin noise.

Brute force tiling of a patch will result in a lot of geometry being drawn, so a level-of-detail (LOD) scheme such as geomipmapping or a quadtree (Lanza 2004) could be employed to reduce the resolution of patches further away from the viewer. Other solutions include a radial grid, as used in the game *Pacific Fighters* (Kryachko 2005), and a projected grid, which contains evenly-spaced vertices in post-perspective camera space, as opposed to world space (Johanson 2004).

The application could make better use of the GPU. For instance, vertex displacement could be handled in a vertex shader by passing a displacement map down the pipeline from the CPU. Normals could be computed in a pixel shader from the displacement map. DirectX 11 introduces GPGPU functionality in the form of compute shaders, which could be used for computation of the FFT and other tasks such as the update loop for wave heights and displacement.

There was insufficient time in this project to tackle other effects like foam and whitecap generation, which become particularly noticeable at Beaufort force 5 and above. When the sea is particularly rough, a particle system could be used to generate spray. Tessendorf (2001) suggests using the Jacobian to detect when to emit foam and

spray. Also, the project does not tackle shallow water effects like breaking waves and caustics, or underwater effects like god rays. Finally, Tessendorf's method only handles ambient waves, so an extension would need to be added in order to provide interactivity, such as in Cords and Staadt (2009) and Yuksel (2010).

Earlier this year, researchers at MIT discovered a more effective sparse FFT that can considerably outperform FFTW in certain scenarios (Hassanieh et al. 2012). The source code of this was recently made public, so an interesting future exercise would be to compare its performance to FFTW in this scenario. A GPU implementation would also be a worthwhile project, and could have a significant impact on the overall performance of the algorithm.

Appendix A. Project Proposal

A.1 Introduction

Water (H_2O) covers roughly 71% of the Earth's surface, and is critical to all known forms of life – indeed, we ourselves are composed predominantly of water. From municipal plumbing to food processing, agriculture to sport, it is ubiquitous in our everyday lives, and as such, is studied across a range of fields. One of these fields is computer graphics, where simulating the visual world around us as closely as possible is often a key goal (Foley et al. 1995).

Fluid simulation can lead to some of the most striking animations in computer graphics, from water splashing to smoke and fire swirling. This project will focus on water – specifically, waves on an ocean surface, as depicted in Figure A.1.

Figure 8: Surface waves on an ocean in mild conditions.

The ocean exhibits highly dynamic behaviour, from minor turbulent waves to enormous shore breaks. The state of the ocean is altered by a multitude of phenomena

occurring at small and large scales. In oceanographic literature, the behaviour of the ocean's surface is generally categorised into two locations: deep areas – those far from the coast – and intermediate or shallow areas – those near to the shore (Darles et al. 2011). A key difference is that in deep water, the effect of the ocean floor is negligible, so wave shoaling and refraction can be ignored.

A.1.1 Aim

The aim of this work is to investigate how waves behave on deep water as they are blown by a light wind. Consideration will be given as to how this complex process can be simplified for real-time processing without overly compromising the essence of what makes the ocean appear as it does. Given the scale of forces at work, this is a challenging problem and still an open area of research in computer animation.

A.1.2 Research Question

How can procedurally animated, realistically lit ocean surface waves be simulated in a real-time environment as they pass across deep water?

A.1.3 Objectives

- To research and detail current approaches to ocean water simulation.
- To construct an application that implements the research discoveries and outputs crisp quantitative results.
- To determine the effectiveness of the implementation by analysis and evaluation of the results.
- To draw conclusions and make recommendations for future work.

A.2 Background

It is expected that the reader is comfortable with classical mechanics and basic vector calculus like the differential operators gradient ∇, divergence $\nabla \cdot$ and curl $\nabla \times$.

A.2.1 Incompressible Navier-Stokes Equations

Most fluid flow in computer graphics is governed by the incompressible Navier-Stokes equations. These are a collection of partial differential equations that should hold throughout the body of the fluid, and are commonly expressed as:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho} \nabla p = \boldsymbol{g} + \upsilon \nabla \cdot \nabla \boldsymbol{u}, \qquad (1)$$

$$\nabla \cdot \boldsymbol{u} = 0. \qquad (2)$$

Where $\vec{u}$ is the velocity of the fluid; ρ is density; p is pressure, the force per unit area exerted by the fluid on something; $\vec{g}$ is acceleration caused by gravity and υ is kinematic viscosity, a measure of how much the fluid resists deformation as it flows (or, as Bridson (2008) explains: "how difficult it is to stir").

Equation 1 is known as the momentum equation, and is derived from Newton's 2nd law of motion, $\vec{F} = m\vec{a}$. It gives us the acceleration of a fluid due to the forces acting on it. Equation 2 is the incompressibility condition. It specifies that the volume of the fluid will remain constant. Changes to volume – such as those caused by underwater acoustics – are usually negligible, and not worth considering in computer graphics, where appearance is the main consideration, not accuracy.

A.2.2 Eulerian and Lagrangian Viewpoints

There are two frames of reference in fluid dynamics: Eulerian and Lagrangian. The Eulerian viewpoint considers how values like velocity and density change with time at fixed points in space. This can be visualized by sitting on a river bank and watching the water flow past a fixed location. The Lagrangian viewpoint, on the other hand, tracks individual fluid parcels as they move through space and time. This is synonymous to the concept of a particle system, and is more in accordance to the molecular composition of fluid in nature. It can be visualized by sitting in a boat as it drifts down a river. In computer graphics, Eulerian fluid is generally arranged into heightmaps or voxels, and Lagrangian fluid is often represented by an implicit surface with particles for control points.

A.2.3 Physically-based and Oceanographic Models

Within the sub-discipline of fluid simulation concerned with oceans, techniques can be categorised as physically-based – those governed by the incompressible Navier-Stokes equations, suitable for shallow water – or spatial/spectral – those based on oceanographic research, suitable for deep water. Spatial methods represent an ocean surface as a heightmap composed of sums of periodical functions, where animation is produced by taking phase differentials. Spectral methods represent an ocean surface as a wave spectrum and apply a Fourier Transform to convert it from the spectral domain to the spatial domain. Hybrid models that combine spatial and spectral characteristics also exist.

A.3 Literature Review

Early exploration into fluid simulation for computer graphics began in the 1980's. Initial methods applied bump mapping to perturb normals in different ways, such as Schachter (1980), who transformed normals using a sum of twenty cycloids. Max (1981) introduced a hydrodynamic model that computed a heightmap from a series of low amplitude sine waves. The frequencies were carefully selected to repeat at a common point in time so that the animation could be looped seamlessly. However, this created a caveat in that the pattern of waves generated could be seen to repeat itself when viewed from afar. In both of these methods, the ocean floor is treated as infinite, so wave shoaling and refraction were not possible as this point in time.

At the ACM SIGGRAPH conference in 1986, two new papers were accepted on ocean water simulation. Peachey (1986) applied Airy wave theory to compute waves that took depth into account for the first time, which meant that breaking waves and refraction could now be simulated. He also used depth to conditionally emit particles for spray, and to apply a quadratic function to basic sinusoids, which gave them a more realistic choppy appearance. In the other paper, Fournier and Reeves (1986) proposed various modifications to Gerstner's wave theory that considered the topology of the ocean floor when calculating the path of water particles, and altered their normal circular path to be elliptic. This meant that the shape of the wave crest could be modified in order to potentially produce a more realistic result.

The first physically-based approach to ocean water simulation was suggested by Kass and Miller (1990). Their method involved approximating the shallow water – or Saint-Venant – equations in two dimensions, and applying the result to a heightmap. Other

simplifications were suggested such as ignoring the vertical component of water particle velocity and treating horizontal velocity in heightmap columns as roughly constant. Despite these simplifications, the method was able to consider wave refraction with depth, reflecting waves, net water transport and scenarios where boundaries vary over time. Chen and Lobo (1995) later extended this approach to solve the Navier-Stokes equations with a pressure term, thus providing a more realistic model of the ocean.

A seminal paper was later written by Stam (1999) in which he introduced an unconditionally stable technique. This turned out to be the key to simulating complex fluid-like behaviour in real-time, as with stability, larger time steps can be taken. This meant that more efficient animation was possible with comparable realism to unstable schemes. The method of Foster and Metaxas (1996) was used as a starting point, but instead of being an explicit Eulerian solver, a combination of Lagrangian and implicit methods was proposed. This approach was later implemented on the GPU by Harris (2004). However, Stam's method is not applicable to this project as it does not address free surfaces like the ocean, and the animation produced is more akin to the movement of gaseous phenomena (Iglesias 2004).

The key paper that this project will seek to investigate was written by Tessendorf (2001), and is based on the work of Mastin, Watterberg and Mareda (1987) who introduced the first spectral approach to ocean simulation. In it, he describes an Eulerian method that utilises statistics from oceanographic research. The ocean is considered incompressible, irrotational (i.e. curl is zero) and inviscid (i.e. no viscosity). Rather than solving the full non-linear Navier-Stokes equations, motion of

the surface is restricted to potential flow, which permits the Navier-Stokes equations to be simplified into Bernoulli's equation. Wave height is considered an arbitrary variable of horizontal position and time. The corresponding heightmap is decomposed into a series of sinusoidal waves and evaluated with an inverse Fast Fourier Transform (FFT). Random sets of amplitudes are created in a way that adheres to oceanographic phenomenology, and the Phillips spectrum is used to model the effect of wind as it blows the waves. A variety of enhancements are proposed to the Phillips spectrum that permit the direction, speed and parallel disposition of the resulting waves to be tuned. Tessendorf also suggests post-processing the heightmap to increase the choppiness of waves, since the inverse FFT does not produce trochoidal shapes.

With regard to industrial application of Tessendorf's method, there have been a few papers written by game developers who are actively using the technique. Mittring (2007) describes how Crytek applied LOD and screen-space tessellation to their implementation of Tessendorf's method in *CryEngine 2*. Day (2009) writes that Insomniac used the technique to simulate water in *Resistance 2*. The technique has also been used significantly within the special effects industry, but via a more accurate approach. The key to the technique's scalability is that the size of the heightmap can be altered. Within the games industry, Insomniac used 32 x 32 heightmaps, and Crytek, 64 x 64, which is in contrast to the 2056 x 2056 heightmaps used for the offline simulations in movies like *Waterworld* and *Titanic* (Tessendorf 2001).

Recently in the field, Yuksel (2010) introduced a new technique called "wave particles" that permits fast, unconditionally stable solid-fluid coupling by tracking surface deviations from waves. This method can be applied to a large-scale body like

an ocean with reasonable performance for a number of interactions. An alternative to this is to solve the 2D wave equation using finite differences, as proposed by Cords and Staadt (2009). The finite differences approach scales better than wave particles when there are lots disturbances because the grid will have a constant number of squares to evaluate at each cycle, regardless of wave activity, whereas wave particles would need to be spawned for each disturbance. Using the wave equation also permits Huygen's principle, which means that waves can be diffracted and the richness of the animation consequently improved. Tessendorf's method is used in both articles to generate ambient (i.e. non-interactive) waves.

A.4 Methodology

A.4.1 Research Design

Fluid dynamics is a broad subject with a significant amount of books, papers and journal articles dedicated to it. There is no universal approach to modelling the various scales of fluid in computer graphics: simulating the movement of a droplet is a different challenge to simulating a river (although many approaches are ultimately governed by the Navier-Stokes equations). As such, it is essential that the project be narrowed down to tackle a particular scenario.

It was decided to investigate Tessendorf's method because his work is frequently cited in other literature, and often lauded as a seminal paper in ocean water simulation. As discovered in the literature review, the technique has seen industrial application in the game and special effects industries, and is still used within recent papers in the field. The fact that it can be applied alongside solid-fluid coupling to create an interactive ocean is particularly relevant to its potential future proliferation in games.

The final application produced will animate a scene like the one in Figure A.1. The user will be able to configure wind speed and direction, and move the camera around the x-z plane for an overhead perspective of the waves. The only geometry present will be the ocean surface and a skybox.

A.4.2 Strategy and Framework

Please refer to the Gantt chart in section A.7 for the intended schedule over the next semester. In it, the project is split into two main deliverables: the application and dissertation. These are further segmented into a series of tasks, each with an expected start date and amount of time required to complete.

Whilst the project focusses on a specific technique, there is still some room for investigation in the approach. For instance, one experiment may contrast various spectrums for animating the waves. Tessendorf (2001) recommends a modified version of the Phillips spectrum, but there are alternatives like the Miles, Pierson-Moskowitz, JONSWAP and TMA spectrums. Additionally, implementing a FFT that is optimised for modern hardware is a significant area of research in itself, and outside the scope of this project. As such, existing implementations will be evaluated for use by the application.

A.4.2.1 Extensions

The Gantt chart also allocates a period of time for extensions. In order of preference, the following tasks may also be undertaken if there is time:

- Add environment mapping to the water surface.
- Add specular reflections from the sun.
- Incorporate objects into the scene, such as islands and rocks that occupy parts of the heightmap, or something that is buoyant.
- Emit particles under certain conditions to represent spray.
- Investigate hardware tessellation to see whether it could speed up displacement mapping.

These would all be of interest when applying the approach to a game, and will be given further consideration as future work if there is not time to implement them.

A.4.2.2 Development Environment

All experiments and profiling will be undertaken on a laptop with the following specifications:

- Intel Core i7 2670 QM, 2.2 GHz
- 6 GB of RAM
- NVIDIA GeForce GT 555M
- Windows 7

The application will be developed in C++ using Visual Studio 2010 and the Direct3D 11 graphics API. It was decided to use Direct3D for the framework because it is the standard graphics API for PC games. CUDA was also considered, but it does not seem to have become widely adopted by the games industry, and is only compatible with

NVIDIA GPUs. Perforce will be used for version control, and to prevent loss of work, workspace captures will be backed up nightly to a cloud server.

A.4.2.3 Resource Requirements

The project does not have any unusual resource requirements. Books, white papers and journal articles will be sourced from the university library. The above development environment is already available, so this section of the project is not considered to be at risk in any way.

A.4.3 Data Collection and Analysis

The two main factors under consideration are performance (quantitative) and realism (qualitative). In order to address the research question, research and experimentation will be undertaken with a view to answering the following questions:

- What are the various techniques for simulating and rendering an ocean in a real-time environment?
- How can Tessendorf's method be efficiently implemented?
- Is the resultant simulation realistic?
- How suitable is it for use within a game?

Specific elements of the model will be profiled with NVIDIA Parallel Nsight in order to locate bottlenecks in the code or compare the run-time performances of two implementations. To gain some insight into how realistic the ocean surface is, the final animation will be juxtaposed to video footage of waves on a real-life ocean under

similar wind conditions. Obviously the degree of realism is a subjective matter, but a critical analysis will be made, and the reader is at liberty to form their own opinion.

A.5 Summary

As established in this document, fluid simulation is an active area of research in the field of computer graphics, with numerous conference papers on the subject appearing each year. Simulating fluid in a game can not only enhance visual fidelity, but with some extension to handle solid-fluid coupling, could also introduce new gameplay mechanics such as the naval warfare in *Empire: Total War*. Additionally, with a configurable wind model, a designer could alter parameters like wind speed to emphasise a dramatic plot line. These are emerging technologies at the moment, but they have the potential to unlock new gameplay experiences and create increasingly immersive virtual worlds. It is hoped that the work undertaken during this project will benefit those looking to simulate ocean water in the future.

Appendix B. Resources CD

B.1 Contents

Dissertation\	An electronic copy of this dissertation in PDF format.
References\	Electronic copies (where available) of reference material.
Software\	Binaries and source code for the application.

Table 5: Directories found on the accompanying resources CD.

B.2 System Requirements

To run the application, a PC with Windows Vista and DirectX 10-compatible GPU capable of executing Shader Model 4 instructions is required. Additionally, Microsoft DirectX SDK (June 2010) should be installed.

Appendix C. Ocean Settings

The application parses configurations from an XML file in order to permit changes without having to recompile any code. The following default settings are applied to the ocean surface by the application:

```
<Ocean>
    <!-- Dimensions of the 2D Fast Fourier Transform (M, N) -->
    <FFTDim>256</FFTDim>

    <!-- Dimensions of the heightmap -->
    <HeightmapDim>128</HeightmapDim>

    <!-- Length of this patch of ocean (Lx, Ly), in metres -->
    <PatchLength>100</PatchLength>

    <!-- Wind direction in azimuth degrees - i.e. N = 0, 360, etc. -->
    <WindDir>200.0f</WindDir>

    <!-- Wind velocity, mph -->
    <WindSpeed>2.0f</WindSpeed>

    <!-- Modifier for global wave amplitude -->
    <WaveAmplitude>0.0000005f</WaveAmplitude>

    <!-- Filter for waves moving opposite to the wind -->
    <WaveFilter>1.00f</WaveFilter>

    <!-- Amount of choppiness to apply to waves -->
    <WaveChoppiness>0.01f</WaveChoppiness>

    <!-- Time interval between consecutive crests at a stationary point -->
    <WavePeriod>2.0f</WavePeriod>

    <!-- Smallest possible wave to arise from wind -->
    <SmallestWave>100000.0f</SmallestWave>
</Ocean>
```

References

Bridson, R. 2008. *Fluid simulation for computer graphics.* India: A K Peters.

Chen, J.X. and Lobo, N.V. 1995. Toward interactive-rate simulation of fluids with moving obstacles using Navier–Stokes equations. *Graphical Models and Image Processing.* 57(2): pp.107–116.

Cooley, J.W. and Tukey, J.W. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation.* 19(1): pp.297-301.

Cords, H. and Staadt, O. 2009. Real-time open water environments with interacting objects. In: E. Galin and J. Schneider, eds. *Eurographics Workshop on Natural Phenomena, Munich, Germany, 1 April 2009.*

Darles, E. et al. 2011. A survey of ocean simulation and rendering techniques in computer graphics. *Computer Graphics Forum.* 30(1): pp.43-60.

Day, M. 2009. *Insomniac's water rendering system.* [online]. Available from: http://www.insomniacgames.com/tech/articles/0409/files/water.pdf [Accessed 30 April 2012]

Foley, J.D. et al. 1995. The quest for visual realism. In: *Computer graphics: principles and practice in C.* 2nd ed. United States of America: Addison-Wesley. 1995, pp.605-649.

Foster, N. and Metaxas, D. 1996. Realistic animation of liquids. In: W.A. Davis and R.M. Bartels, eds. *Proceedings of Graphics Interface 1996, Calgary, Canada, 22-24 May 1996.* pp.204–212.

Fournier, A. and Reeves, W.T. 1986. A simple model of ocean waves. In: *SIGGRAPH 1986 Conference Proceedings, Dallas, Texas, 18-22 August 1986.* New York: ACM Press/Addison-Wesley. 20(4): pp.75–84.

French, A.P. 1971. *Newtonian mechanics.* Great Britain: W.W. Norton & Company.

Gomez, M. 2000. Interactive simulation of water surfaces. In: M. DeLoura, ed. *Game programming gems.* United States of America: Charles River Media. 2000, pp.187-194.

Harris, M. 2004. Fast fluid dynamics simulation on the GPU. In: R. Fernando, ed. *GPU gems: programming techniques, tips, and tricks for real-time graphics.* United States of America: Addison-Wesley. 2004, pp.637-665.

Hassanieh, H. et al. 2012. Simple and practical algorithm for sparse Fourier transform. In: *ACM-SIAM Symposium On Discrete Algorithms, Kyoto, Japan, 17-19 January 2012.*

Iglesias, A. 2004. Computer graphics for water modeling and rendering: a survey. *Future Generation Computer Systems.* 20(8): pp.1355-1375.

Jensen, L.S. and Goliáš, R. 2001. *Deep-water animation and rendering.* [online]. Available from: http://www.gamasutra.com/gdce/2001/jensen/jensen_01.htm [Accessed 1 May 2012]

Johanson, C. 2004. *Real-time water rendering: introducing the projected grid concept.* [MSc thesis]. Lund University.

Kass, M. and Miller, G. 1990. Rapid, stable fluid dynamics for computer graphics. In: *SIGGRAPH 1990 Conference Proceedings, Dallas, Texas, 6-10 August 1990.* New York: ACM Press/Addison-Wesley. 24(4): pp.49–57.

Kryachko, Y. 2005. Using vertex texture displacement for realistic water rendering. In: M. Pharr, ed. *GPU gems 2: programming techniques for high-performance graphics and general-purpose computation.* United States of America: Addison-Wesley. 2005, pp.283-294.

Lanza, S. 2004. Animation and display of water. In: W. Engel, ed. *ShaderX3: advanced rendering with DirectX and OpenGL.* United States of America: Charles River Media. 2004, pp.215-228.

Losasso, F. et al. 2008. Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics.* 14(4): pp.797-804.

Mastin, G.A., Watterberg, P.A. and Mareda, J.F. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications.* 7(3): pp.16–23.

Matthews, P.C. 1998. *Vector calculus.* Great Britain: Springer-Verlag.

Max, N. 1981. Vectorized procedural models for natural terrain: waves and islands in the sunset. In: *SIGGRAPH 1981 Conference Proceedings, Dallas, Texas, 3-7 August 1981.* New York: ACM Press/Addison-Wesley. pp.317-324.

Mittring, M. 2007. Finding next gen – CryEngine 2. In: *Advanced Real-Time Rendering in 3D Graphics and Games, SIGGRAPH 2007, San Diego, California, 8 August 2007.* pp.97-121.

NASA. [no date]. *Ocean color.* [online]. Available from: http://science.nasa.gov/earth-science/oceanography/living-ocean/ocean-color/ [Accessed 3 May 2012]

Peachey, D.R. 1986. Modeling waves and surf. In: *SIGGRAPH 1986 Conference Proceedings, Dallas, Texas, 18-22 August 1986.* New York: ACM Press/Addison-Wesley. 20(4): pp.65–74.

Schachter, B. 1980. Long crested wave models. *Computer Graphics and Image Processing.* 12(2): pp.187-201.

Schlick, C. 1994. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum.* 13(3): pp.233-246.

Stam, J. 1999. Stable fluids. In: *SIGGRAPH 1999 Conference Proceedings, Los Angeles, California, 8-13 August 1999.* New York: ACM Press/Addison-Wesley. pp.121-128.

Tessendorf, J. 2001. Simulating ocean water. In: *Simulating Nature: Realistic and Interactive Techniques, SIGGRAPH 2001 Course #47 Notes, Los Angeles, California, 12-17 August 2001.*

Thomas, D.B. et al. 2007. Gaussian random number generators. *ACM Computing Surveys.* 39(4): pp.11:1-11:38.

Wikipedia. 2012. *Schlick's approximation.* [online]. Available from: http://en.wikipedia.org/wiki/Schlick's_approximation [Accessed 7 May 2012]

Yuksel, C. 2010. *Real-time water waves with wave particles.* [PhD dissertation]. Texas A&M University.

Bibliography

Akenine-Möller, T., Haines, E. and Hoffman, N. 2008. *Real-time rendering.* 3rd ed. India: A K Peters.

Anderson, J.D. 1995. *Computational fluid dynamics: the basics with applications.* United States of America: McGraw-Hill.

Fernando, R. and Kilgard, M.J. 2003. *The Cg tutorial: the definitive guide to programmable real-time graphics.* United States of America: Addison-Wesley.

Frigo, M. and Johnson, S.G. 2012. *FFTW user manual – version 3.3.2.* [online]. Available from: http://www.fftw.org/fftw3.pdf [Accessed 5 May 2012]

Knauss, J.A. 2005. *Introduction to physical oceanography.* 2nd ed. United States of America: Waveland Press.

Lamb, H. 1932. *Hydrodynamics.* 6th ed. United States of America: Cambridge University Press.

Zink, J., Pettineo, M. and Hoxley, J. 2011. *Practical rendering and computation with Direct3D 11.* United States of America: A K Peters/CRC Press.